```c
#include <stdio.h>
#include <stdlib.h>

int count = 0;
typedef struct Node
{
    int data;
    struct Node* next;
} Node;

Node* createNode(int value)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode)
    {
        printf("Memory allocation failed!\n");
        return NULL;
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

Node* createLinkedList(Node* last)
{
    int value, choice;
    while (1)
    {
        printf("Enter value to insert: ");
        scanf("%d", &value);
        Node* newNode = createNode(value);

        if (!last)
        {
            last = newNode;
            newNode->next = newNode;
        }
        else
        {
            newNode->next = last->next;
            last->next = newNode;
            last = newNode;
        }
        count++;
        printf("Add another node? (1-Yes / 0-No): ");
        scanf("%d", &choice);
        if (choice == 0)
            break;
    }
    printf("Circular linked list created successfully with %d nodes.\n",
count);
    return last;
}

Node* insertAtBeginning(Node* last, int value)
{
    Node* newNode = createNode(value);

    if (!last)
    {
        last = newNode;
        newNode->next = newNode;
    }
    else
```

```
    {
        newNode->next = last->next;
        last->next = newNode;
    }
    count++;
    printf("Element %d inserted at beginning.\n", value);
    return last;
}

Node* insertAtEnd(Node* last, int value)
{
    Node* newNode = createNode(value);
    if (!last)
    {
        last = newNode;
        newNode->next = newNode;
    }
    else
    {
        newNode->next = last->next;
        last->next = newNode;
        last = newNode;
    }
    count++;
    printf("Element %d inserted at end.\n", value);
    return last;
}

Node* insertAtPosition(Node* last, int pos, int value)
{
    if (pos < 1 || pos > count + 1)
    {
        printf("Invalid position! Valid positions: 1 to %d\n", count + 1);
        return last;
    }

    Node* newNode = createNode(value);

    if (pos == 1)
    {
        if (last == NULL)
        {
            last = newNode;
            newNode->next = newNode;
        }
        else
        {
            newNode->next = last->next;
            last->next = newNode;
        }
        count++;
        printf("Element %d inserted at position 1.\n", value);
        return last;
    }

    Node* temp = last->next;
    for (int i = 1; i < pos - 1; i++)
        temp = temp->next;

    newNode->next = temp->next;
    temp->next = newNode;

    if (temp == last)
        last = newNode;
```

```
    count++;
    printf("Element %d inserted at position %d.\n", value, pos);
    return last;
}

Node* insertByOrder(Node* last, int value)
{
    Node* newNode = createNode(value);

    if (!last)
    {
        last = newNode;
        newNode->next = newNode;
        count++;
        return last;
    }

    if (value < last->next->data)
    {
        newNode->next = last->next;
        last->next = newNode;
        count++;
        return last;
    }

    Node* temp = last->next;
    while (temp->next != last->next && temp->next->data < value)
        temp = temp->next;

    newNode->next = temp->next;
    temp->next = newNode;

    if (temp == last)
        last = newNode;

    count++;
    return last;
}

Node* deleteBeginning(Node* last)
{
    if (!last)
    {
        printf("List is empty. Deletion not possible.\n");
        return last;
    }

    Node* first = last->next;

    if (first == last)
    {
        printf("Element %d deleted from beginning.\n", first->data);
        free(first);
        count--;
        return NULL;
    }

    last->next = first->next;
    printf("Element %d deleted from beginning.\n", first->data);
    free(first);
    count--;
    return last;
}
```

```
Node* deleteEnd(Node* last)
{
    if (!last)
    {
        printf("List is empty. Deletion not possible.\n");
        return last;
    }

    Node* first = last->next;

    if (first == last)
    {
        printf("Element %d deleted from end.\n", last->data);
        free(last);
        count--;
        return NULL;
    }

    Node* temp = first;
    while (temp->next != last)
        temp = temp->next;

    temp->next = last->next;
    printf("Element %d deleted from end.\n", last->data);
    free(last);
    count--;
    last = temp;
    return last;
}

Node* deleteAtPosition(Node* last, int pos)
{
    if (!last)
    {
        printf("List is empty. Deletion not possible.\n");
        return last;
    }

    if (pos < 1 || pos > count)
    {
        printf("Invalid position! Deletion not possible.\n");
        return last;
    }

    Node* toDelete;
    if (pos == 1)
    {
        toDelete = last->next;
        if (last == toDelete)
        {
            printf("Element %d deleted from position 1.\n", toDelete->data);
            free(toDelete);
            count--;
            return NULL;
        }
        last->next = toDelete->next;
        printf("Element %d deleted from position 1.\n", toDelete->data);
        free(toDelete);
        count--;
        return last;
    }

    Node* prev = last->next;
```

```
    for (int i = 1; i < pos - 1; i++)
        prev = prev->next;

    toDelete = prev->next;
    prev->next = toDelete->next;

    if (toDelete == last)
        last = prev;

    printf("Element %d deleted from position %d.\n", toDelete->data, pos);
    free(toDelete);
    count--;

    return last;
}

Node* deleteByKey(Node* last, int key)
{
    if (!last)
    {
        printf("List is empty. Deletion not possible.\n");
        return last;
    }

    Node* first = last->next;
    if (first->data == key)
        return deleteBeginning(last);

    Node* temp = first;
    while (temp->next != first && temp->next->data != key)
        temp = temp->next;

    if (temp->next->data != key)
    {
        printf("Key %d not found.\n", key);
        return last;
    }

    Node* toDelete = temp->next;
    temp->next = toDelete->next;

    if (toDelete == last)
        last = temp;

    printf("Key %d deleted from the list.\n", key);
    free(toDelete);
    count--;
    return last;
}

Node* searchByKey(Node* last, int key)
{
    if (!last)
    {
        printf("The list is empty. Cannot search.\n");
        return NULL;
    }

    Node* first = last->next,* temp = first;
        do
    {
        if (temp->data == key)
        {
            printf("Key %d found in the list.\n", key);
```

```
            return temp;
        }
        temp = temp->next;
    } while (temp != first);

    printf("Key %d NOT found in the list.\n", key);
    return NULL;
}

Node* reverseList(Node* last)
{
    if (!last || last->next == last)
    {
        printf("List is empty or has only one node. Nothing to reverse.\n");
        return last;
    }

    Node *prev = last, *current = last->next, *nextNode;
    Node* first = last->next;

    do
    {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    } while (current != first);

    last = first;
    printf("List has been reversed successfully.\n");
    return last;
}

Node* copyList(Node* last)
{
    if (!last)
        return NULL;

    Node* newFirst = NULL;
    Node* newLast = NULL;
    Node* current = last->next;

    do
    {
        Node* newNode = createNode(current->data);

        if (!newFirst)
        {
            newFirst = newNode;
            newLast = newNode;
            newNode->next = newNode;
        }
        else
        {
            newLast->next = newNode;
            newNode->next = newFirst;
            newLast = newNode;
        }

        current = current->next;
    } while (current != last->next);

    printf("List has been copied successfully.\n");
    return newLast;
```

```
}

void printList(Node* last)
{
    if (!last)
    {
        printf("List is empty.\n");
        return;
    }

    Node* first = last->next;
    Node* temp = first;

    printf("Current list (%d nodes): ", count);
    do
    {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != first);
    printf("\n");
}

int main()
{
    Node* last = NULL;
    int choice, value, pos, key;

    while (1)
    {
        printf("\n--- Circular Singly Linked List Menu ---\n");
        printf("1. Create List\n2. Insert at Beginning\n3. Insert at End\n");
        printf("4. Insert at Position\n5. Insert By Order\n6. Delete from
Beginning\n7. Delete from End\n");
        printf("8. Delete from Position\n9. Delete By Key\n10. Search by
Key\n11. Reverse List\n12. Copy List\n13. Print List\n14. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1: last = createLinkedList(last); break;
            case 2: printf("Enter value: "); scanf("%d", &value); last =
insertAtBeginning(last, value); break;
            case 3: printf("Enter value: "); scanf("%d", &value); last =
insertAtEnd(last, value); break;
            case 4: printf("Enter position: "); scanf("%d", &pos);
                    printf("Enter value: "); scanf("%d", &value);
                    last = insertAtPosition(last, pos, value);
                    break;
            case 5: printf("Enter value: "); scanf("%d", &value); last =
insertByOrder(last, value); break;
            case 6: last = deleteBeginning(last); break;
            case 7: last = deleteEnd(last); break;
            case 8: printf("Enter position: "); scanf("%d", &pos); last =
deleteAtPosition(last, pos); break;
            case 9: printf("Enter key: "); scanf("%d", &key); last =
deleteByKey(last, key); break;
            case 10: printf("Enter key to search: "); scanf("%d", &key);
searchByKey(last, key); break;
            case 11: last=reverseList(last); break;
            case 12: last=copyList(last); break;
            case 13: printList(last); break;
            case 14: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
```

```
        }
    }
    return 0;
}
```